Reconfiguration dynamique du placement dans les grilles de calculs dirigée par des objectifs

Fabien Hermenier, Xavier Lorca, Hadrien Cambazard, Jean-Marc Menaud, Narendra Jussien

RenPar'18 / SympA'2008 / CFSE'6
Fribourg, Suisse, du 11 au 13 février 2008

Reconfiguration dynamique du placement dans les grilles de calcul dirigée par des contraintes

Fabien Hermenier,[1] Xavier Lorca,[1] Hadrien Cambazard,[2] Jean-Marc Menaud,[1] Narendra Jussien[1]

[1]Departement Informatique, Ecole des Mines de Nantes – INRIA, LINA
4, rue Alfred Kastler – 44307 Nantes Cedex 3, France
{fhermeni,xlorca,menaud,jussien}@emn.fr

[2]Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
h.cambazard@4c.ucc.ie

Résumé
L'utilisation des grilles de calcul par un ensemble toujours plus important de personnes a rendu la problématique de l'optimisation du placement des applications délicate. Les administrateurs sont en effet perpétuellement confrontés à des contraintes de placement qui doivent être impérativement satisfaites. Ces contraintes portent sur des ensembles précis d'applications ou sur une portion de l'architecture de la grille et rendent donc l'utilisation d'une seule politique globale inefficace. Simultanément, les besoins en ressources, tout comme leur disponibilité varient d'une manière importante au cours du temps. Une approche statique du placement guidée par des objectifs d'optimisation fixes n'est donc plus valable.
Nous proposons dans cet article une architecture flexible permettant d'adapter le placement des applications en fonction d'une analyse des besoins, de l'état courant des ressources et de contraintes de placement définies par l'utilisateur. Le développement et l'évaluation d'un ensemble de contraintes assurant une répartition des ressources adaptées au besoin des applications et maintenant un nombre de nœuds utiles minimum nous a permis de valider expérimentalement notre approche.

Mots-clés : grilles, placement, contraintes

1. Introduction

Les grilles de calcul sont une fédération de ressources informatiques hétérogènes (ordinateurs personnels, grappe de calcul, supercalculateurs...), géographiquement éloignées et mises en réseau [11]. Cette technologie permet de mettre la puissance de calcul et la capacité de stockage de milliers d'ordinateurs en commun et de fournir aux utilisateurs d'une grille d'immenses capacités informatiques. Depuis quelques années, les grilles de calcul ont réalisé des progrès majeurs et ont atteint une maturité telle qu'elles constituent actuellement un moyen de calcul extrêmement puissant au service de nombreuses organisations. Parmi ces grands systèmes de calcul distribués, la grille EGEE [1] constitue ainsi une véritable infrastructure mondiale de production, traitant 24 heures sur 24 plusieurs millions de travaux mensuels pour des milliers d'utilisateurs provenant de plus d'une dizaine de disciplines scientifiques. La complexité de ces grilles ne cesse de croître d'autant plus que nous parlons maintenant de fédération de grilles, comme celle utilisée pour le projet Wisdom. Grâce à l'association de plusieurs grilles de calcul internationales, dont la grille européenne EGEE, le projet Wisdom a permis d'analyser près de 80 000 médicaments potentiels par heure pendant 10 semaines, dans le cadre du traitement de la malaria.
La complexité de gestion de ces grilles tient autant à l'hétérogénéité des matériels agrégés, qu'aux spécificités techniques des expérimentations propres aux organisations où encore aux politiques des administrateurs concernant la gestion des éléments formant la grille.
La multiplicité des protagonistes agissant sur les grilles en fait une structure difficilement administrable, avec pour conséquence, des limites concernant les possibilités d'optimisation de son utilisation.
Pour exemple, le cluster administré par l'École des Mines de Nantes dans le cadre de la grille de l'IN2P3, est subdivisé en deux parties isolées : les machines dédiées au calculs propres à l'IN2P3 et les machines dédiées

aux calculs des physiciens locaux. Cette subdivision est définie statiquement, si bien que la puissance de calcul disponible dans une des parties (en cas de sous-utilisation) ne peut être offerte à l'autre partie. Cette contrainte de partitionnement des ressources est propre à ce cluster et limite son potentiel d'utilisation pour les deux parties et donc le taux d'utilisation global de la grille. Cet exemple n'est pas isolé puisque le Parallel Workloads Archive [23] qui récolte des traces d'activités de 19 clusters montre un taux d'utilisation de l'ordre de 49,96%. Cependant, si ce taux moyen est effectivement très bas, il faut noter que des pics d'activités consommant la totalité des ressources disponibles sont observables. Dans une optique d'économie d'énergie, il conviendrait d'éteindre les machines non-utilisées. Seulement avec les variations d'activité observable, un partitionnement statique des nœuds à utiliser n'est donc pas possible. Il convient de redéfinir cette politique dynamiquement. Cet exemple peut être étendu également au niveau des utilisateurs qui possèdent généralement des contraintes relatives au placement de leurs applications.
Au travers de ces exemples, la complexité d'administration et d'optimisation des grilles de calcul est mise en évidence et il devient nécessaire de disposer d'un système autonome et dynamique configurable. Notre contribution porte sur la définition et l'évaluation d'un système d'optimisation autonome des grilles de calcul, la conception d'un algorithme de placement dynamique minimisant les migrations de calcul entre machines, et l'évaluation théorique et pratique de la solution mise en place.
La suite de cet article se consacre tout d'abord sur la motivation de notre problématique. En section 3 nous présentons l'architecture de notre prototype. La section 4 présente une évaluation de celui-ci, d'après une comparaison théorique d'algorithmes et d'un résumé d'exécution basé sur des traces réelles de clusters. La section 5 décrit les travaux s'apparentant à notre problématique. Finalement, nous concluons en section 6.

2. Motivations

Notre objectif est de développer un système autonome d'optimisation des ressources pour grille de calcul. La gestion des ressources consiste à placer, statiquement (lors de la création du processus) ou dynamiquement (par migration du calcul), les calculs effectués par les utilisateurs de la grille de manière optimale par rapport aux besoins des utilisateurs. Ce placement est guidé à la fois par l'état actuel des ressources mais également par des contraintes de placement.
De part leur taille, les grilles informatiques sont soumises à un grand niveau d'hétérogénéité. En effet, d'un point de vue matériel, chaque cluster composant la grille est acheté par des administrations différentes. Chacune de ces administrations réalise son propre appel d'offres et sélectionne les meilleurs constructeurs en fonction de critères qui lui sont propres. Par conséquent, l'hétérogénéité matérielle est inévitable. Ce problème se pose également concernant l'environnement logiciel. Chaque organisme ou groupement d'organismes, regroupés sous l'appellation d'organisation virtuelle, réalisent des calculs spécifiques à leur thème scientifique. Par conséquent, chaque organisation virtuelle requiert des besoins logiciels spécifiques. Pour exemple, l'application Aliroot utilisée par l'IN2P3 nécessite en plus d'un intergiciel particulier (Alien), une version spécifique de Linux (Scientifique Linux). Il est alors illusoire d'imposer à toutes les organisations un même environnement logiciel d'exécution et irréaliste d'en concevoir un répondant à toutes les spécificités des expérimentations. Un service de reconfiguration pour grille doit donc pour pouvoir s'adapter à la fois à la diversité du matériel et à la spécificité des besoins des organisations virtuelles.
Une solution pour ces problèmes consiste à séparer la partie applicative et la partie reconfiguration en définissant une nouvelle couche d'abstraction. Cette couche a pour objectif de masquer l'hétérogénéité de la grille tout en mettant à disposition un ensemble de mécanismes permettant la supervision et la manipulation des environnements utilisateurs. Le service est alors à la fois générique, et non-intrusif. La virtualisation est une solution efficace permettant d'implémenter cette couche d'abstraction [2, 10, 21, 16]. Dans cette situation, chaque organisation virtuelle dispose de son propre ensemble de machines virtuelles, placées sur des noeuds selon une configuration précise. De plus, grâce aux capacités de migration à chaud des intergiciels de bas niveau, comme Xen [4] ou KVM, il est possible de placer et déplacer dynamiquement les calculs sans interruption de ces derniers en tenant compte d'un certains nombre de critères de placement.

La définition de ces critères dépend à la fois de l'architecture de la grille mais également des différentes applications s'exécutant dessus. On peut distinguer deux types d'acteurs dans les grilles : les administrateurs et les utilisateurs. Chacun des acteurs est susceptible de définir un ensemble de contraintes à satisfaire. Les contraintes des administrateurs portent sur l'architecture physique de la grille et l'agencement global des machines virtuelles

(en vue d'économiser l'énergie par exemple). Les contraintes définies par les utilisateurs servent le plus souvent à assurer la compatibilité de leurs applications avec leurs hôtes, mais également à définir une certaine qualité de services assurant la disponibilité de certaines ressources.
Devant cette variété, il importe alors de définir un système de reconfiguration flexible, où les différents acteurs de la grille peuvent brancher et combiner les différentes contraintes qu'ils souhaitent voir satisfaite. Le moteur doit alors pouvoir trouver les nouvelles configurations compatibles avec toutes les contraintes suivant l'état des ressources de la grille et des besoins des applications.
Entropy, notre solution, repose sur une architecture virtualisée, appelée *système vertébral* et propose un système flexible de placement regroupé dans le *système cérébral*. Ces deux sous-systèmes sont détaillés dans la suite de cet article.

3. Architecture d'Entropy

Entropy permet de reconfigurer à la volée le placement des environnements utilisateurs en maintenant un ensemble de contraintes. Le processus de reconfiguration est articulé autour d'une boucle de contrôle autonomique [14] (voir figure 1) : d'après les informations remontées depuis différentes sondes, un solveur de contraintes propose, d'après un ensemble de critères mis en place par les utilisateurs un ensemble de solutions améliorant le placement des applications.

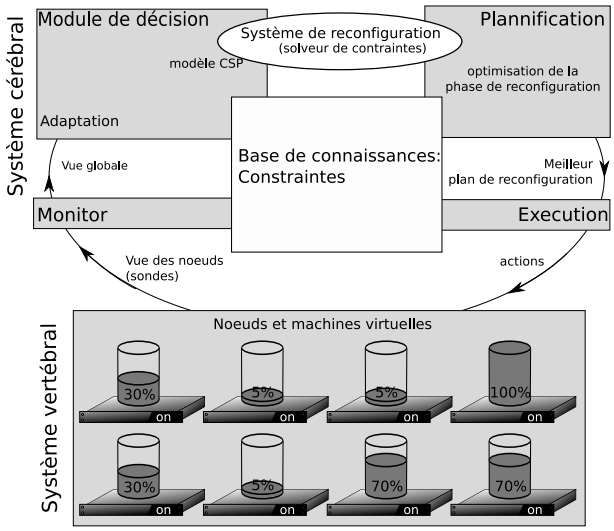


FIG. 1 – Boucle de contrôle d'Entropy

3.1. Le système vertébral : la couche d'abstraction

Les besoins des utilisateurs pouvant être très spécifiques en terme de matériel, système d'exploitation ou intergiciel, le service se doit d'être effectif tout en étant transparent pour l'utilisateur. Il convient alors de séparer dans différentes couches les aspects d'utilisation de la grille et reconfiguration de celle-ci.
La virtualisation des grilles est une solution efficace permettant de séparer ces préoccupations. Le but de la virtualisation est de proposer à chaque utilisateur l'illusion de disposer de ses propres machines (virtuelles), mais qui en réalité s'exécutent en parallèle sur une ou plusieurs machines réelles. Dans un environnement virtualisé, un *hyperviseur*, s'intercalant entre la couche matérielle et les machines virtuelles permet la gestion et le partage de ressources entre celles-ci.
La virtualisation permet donc à un hôte d'héberger de manière sûre plusieurs machines virtuelles isolées. Chaque machine virtuelle peut alors disposer de son propre système d'exploitation, intergiciel ou applications, sans aucun risque de conflit avec les environnements de gestion de la grille ou utilisateurs. Cette architecture permet également, par l'ajout d'une couche d'abstraction de bas niveau de contrôler de façon transparente et générique

les environnements utilisateurs. Il est alors possible de superviser la distribution des ressources, ou de migrer ces machines virtuelles sans arrêt de service [8] et sans intrusion dans l'espace utilisateur.
Entropy utilise l'hyperviseur Xen [4]. La supervision est assurée par un ensemble de sondes installées sur chaque noeud. Celles-ci interrogent régulièrement chaque hyperviseur afin observer la distribution des ressources entre les différentes machines virtuelles et l'état de chaque nœud. Analyser les environnements utilisateurs au niveau de l'hyperviseur est une solution générique et non intrusive permettant d'obtenir la consommation et les besoins en ressources de chaque machine virtuelle. Il n'est donc plus nécessaire de modifier ou de porter les outils de supervision pour qu'ils s'adaptent aux spécificitées des environnements utilisateurs.

3.2. Le système Cérébral

Le système cérébral est subdivisé en quatre parties. Un ensemble de sondes placées sur les hyperviseurs de chaque noeud permet de faire remonter l'état des ressources de la grille (taux d'utilisation et distribution des ressources CPU, mémoire). Un module de décision permet ensuite de générer par rapport à l'état de la grille et aux contraintes définies par les acteurs un ensemble de nouvelles configurations optimisant le placement. Une phase d'optimisation est chargée ensuite de sélectionner la meilleure configuration tout en assurant une phase de transition sûre. Enfin, une phase d'exécution permet de passer de la configuration courante à la nouvelle.

3.2.1. Calcul des placements : Génération de la nouvelle configuration

L'hétérogénéité des grilles de calcul implique la prise en compte d'un certain nombre de contraintes relatives aux caractéristiques des produits matériels et logiciels mis en jeu. En parallèle de ces contraintes *dures*, les utilisateurs spécifient très souvent de multiples contraintes de placement (vues comme des critères d'optimisation) quant au déploiement des travaux à effectuer.
Il importe alors de pouvoir spécialiser à la volée le gestionnaire de reconfiguration. Nous utilisons pour cela un solveur de contraintes utilisant une approche basée sur la Programmation Par Contraintes[20, 5]. L'idée de La Programmation Par Contraintes (PPC) est de proposer des solutions à un problème en spécifiant seulement un ensemble de contraintes (conditions, propriétés) devant être satisfaites par toute solution acceptable pour le problème donné. Un problème de satisfaction de contraintes est alors modélisé sous la forme d'un ensemble de variables (au sens mathématique), d'un ensemble de domaines représentant les différentes valeurs que peuvent prendre les variables et de contraintes formant un ensemble de relations entre celles-ci. Une solution représente alors une affectation de valeurs à chaque variable satisfaisant simultanément l'ensemble des contraintes. Un solveur de contraintes est un moteur permettant de générer ces solutions, les utilisateurs de la PPC n'ont alors qu'à décrire leurs variables, leurs domaines et leurs contraintes. Il s'agit donc d'un paradigme de programmation déclaratif.
Dans notre cas, la PPC nous apporte une solution algorithmique flexible au problème de la variété des critères exposés précédemment. Les données récupérées durant la phase de supervision sont transformées en un ensemble de variables/domaines. Les différents critères de placement (contraintes dures) et d'optimisation sont mis à disposition depuis une base de connaissances sous la forme de contraintes. Celles-ci sont alors connectées à la volée sur le solveur et forme un problème d'affectation de machines virtuelles à des nœuds (problème comparable au problème de *bin packing multidimensionnel* [22]).
À l'heure actuelle, le modèle d'affectation de machines virtuelles est représenté par un ensemble $\mathcal{N}$ de n noeuds (ou hôtes). Cet ensemble est caractérisé par trois variables, définissant l'architecture système de la machine (constante), sa capacité CPU et la quantité de mémoire vive utilisable par les machines virtuelles. L'ensemble $\mathcal{V}$ des k machines virtuelles est représenté par trois variables définissant son architecture système (constante), ses besoins en CPU (variable) et la mémoire vive qui lui est allouée (constante). La définition suivante représente formellement cette affectation :

Définition 3.1 *Pour chaque nœud* $i \in \mathcal{N}$*, un vecteur booléen* [1] $X_i = \langle x_{i1}, \ldots, x_{ij}, \ldots, x_{ik} \rangle$ *denote l'ensemble des machines virtuelles affectée au nœud* i*, et la variable booléenne*[2] u_i *représente l'affectation du nœud* i *à au moins une machine virtuelle.*

Entropy est basé sur le solveur de contraintes Choco [7]. Ce solveur met à disposition une librairie JAVA permettant la déclaration de problèmes de satisfactions de contraintes. À ce jour, deux jeux de contraintes ont été implémenté

[1] $x_{ij} = 1$ si le nœud i héberge la machine virtuelle j.
[2] $u_i = 1$ si il existe une machine virtuelle $j \in \mathcal{V}$ telle que $x_{ij} = 1$.

en utilisant cette librairie et l'API fournit par Entropy (permettant la manipulation des machines virtuelles). Le premier jeu de contraintes concerne les organisations virtuelles et permet d'assurer dynamiquement à chaque machine virtuelle une quantité de mémoire et de temps CPU au moins égale à leurs besoins (voir définition 3.2).

Définition 3.2

$$\mathcal{D}_c \cdot X_i \leq \mathcal{C}[i] \forall i \in \mathcal{N} \quad (1)$$

$$\mathcal{D}_m \cdot X_i \leq \mathcal{M}[i] \forall i \in \mathcal{N} \quad (2)$$

Où $\mathcal{C}$ est le vecteur des capacités CPU associées à chaque nœud, $\mathcal{M}$ est le vecteur des capacités mémoire associées à chaque nœud, $\mathcal{D}_c$ est le vecteur des besoins CPU de chaque machine virtuelle et $\mathcal{D}_m$ le vecteur des besoins en mémoire de chaque machine virtuelle.

Une contrainte d'optimisation, concernant cette fois les administrateurs de clusters a également été implémentée. Elle permet de minimiser le nombres de nœuds nécessaires à l'hébergement des machines virtuelles en vue de réduire la consommation électrique. En cas d'augmentation des besoins des machines virtuelles, le solveur choisit de nouveaux nœuds à allumer afin d'y assigner des machines virtuelles. Cet objectif a été formalisé de la façon suivante :

Définition 3.3

$$\min \sum_{i \in \mathcal{N}} u_i, \textit{ sujet à l'équation 1 et 2} \quad (3)$$

Où u_i est une variable booléenne évaluée à vrai *si le nœuds i est allumé.*

3.2.2. Optimisation de la reconfiguration : minimisation des migrations

Le moteur de génération de configurations peut donc grâce à la modélisation de l'état de la grille et la définition des critères et contraintes de placement générer un ensemble de nouvelles configurations adaptées à l'état courant de la grille.

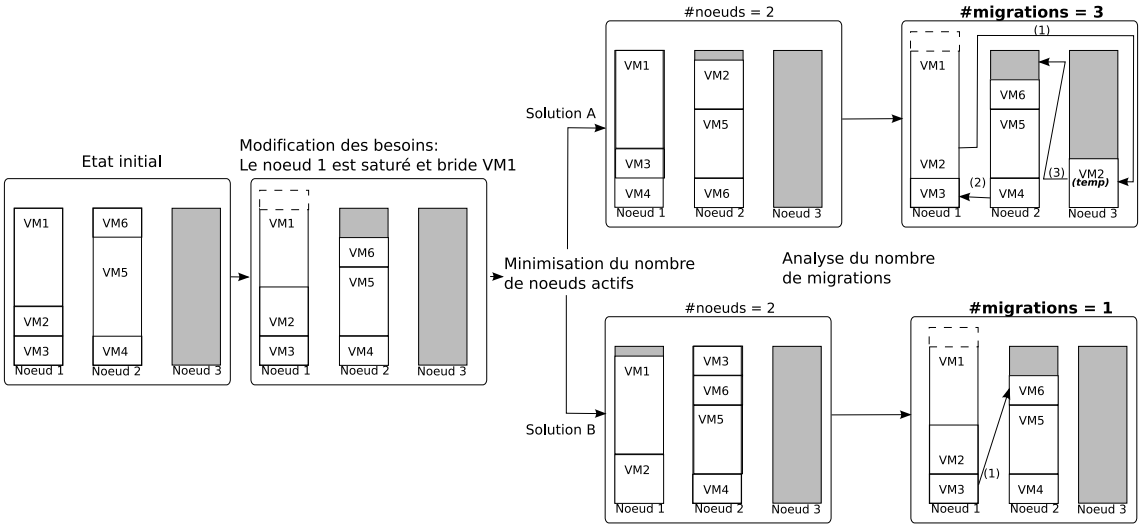


FIG. 2 – Exemple de solutions non-équivalentes

Cependant toutes les nouvelles configurations ainsi produites ne sont pas équivalentes. En effet, dans le contexte d'évolution dynamique des grilles de calcul, le passage d'une configuration courante à une configuration optimisée nécessite très souvent la migration de certaines machines virtuelles (voir figure 2). Dans le cadre d'un système

à l'échelle d'une grille, il est important de tenir compte de ce nombre de migrations à effectuer afin d'assurer une solution réactive. Le nombre de migrations assurant la transition entre deux configurations est donc un critère d'optimisation à prendre en compte.

Cette phase d'optimisation cherche à sélectionner la meilleure configuration parmi les solutions générées par le solveur en se basant sur la comparaison des *plans de reconfiguration*. La création de ces plans permet de repérer et supprimer les migrations critiques afin d'assurer que la grille et les machines virtuelles restent dans un état consistant durant la reconfiguration. Une phase de reconfiguration peut être vue comme un graphe orienté où une migration représente un arc entre deux nœuds (figure 3). Cette représentation nous permet de remarquer la présence de cycles (migrations de $\mathtt{VM}_1$, $\mathtt{VM}_2$ et $\mathtt{VM}_3$ sur le graphe), ou la nécessité d'imposer un séquencement (si N_5 est considéré comme *plein*, nous devons nous assurer de migrer $\mathtt{VM}_6$ ou $\mathtt{VM}_7$ avant de rapatrier $\mathtt{VM}_5$).

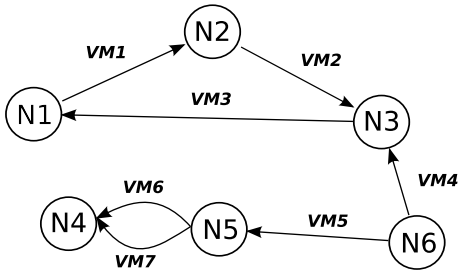


FIG. 3 – Un graphe de reconfiguration

$$
\begin{aligned}
&S_1 := N_1 \xrightarrow{\mathtt{VM}_1} pivot; N_3 \xrightarrow{\mathtt{VM}_3} N_1; N_2 \xrightarrow{\mathtt{VM}_2} N_3;\\
&pivot \xrightarrow{\mathtt{VM}_1} N_2\\
&S_2 := N_6 \xrightarrow{\mathtt{VM}_4} N_3\\
&S_3 := N_5 \xrightarrow{\mathtt{VM}_6} N_4\\
&S_4 := N_5 \xrightarrow{\mathtt{VM}_7} N_4; N_6 \xrightarrow{\mathtt{VM}_5} N_5
\end{aligned}
$$

FIG. 4 – Plan de reconfiguration associé

Rendre la phase de reconfiguration sûre consiste donc à transformer ce graphe en un ensemble de *séries*, imposant un séquencement entre les migrations critiques. Les dépendances entre migrations sont exécutées dans l'ordre inverse de leur apparition (la dernière migration d'une séquence est exécutée en dernier). Les cycles sont transformés en séries en insérant une migration supplémentaire sur un nœud temporaire appelé *pivot*, choisi dynamiquement. L'ensemble de ces séries forme le *plan de reconfiguration* (figure 4).

Une fois le plan de reconfiguration de chaque solution fourni par le solveur, une extension du modèle décrit dans la section précédente, permet de sélectionner le plan de reconfiguration entraînant le moins d'opérations de migrations à réaliser. Lors de l'application du plan de reconfiguration, les différentes séries, indépendantes, sont exécutées en parallèle. À l'intérieur de chaque série, les différentes migrations sont réalisées séquentiellement en envoyant les ordres de migrations aux hyperviseurs des nœuds concernés.

4. Évaluation d'Entropy

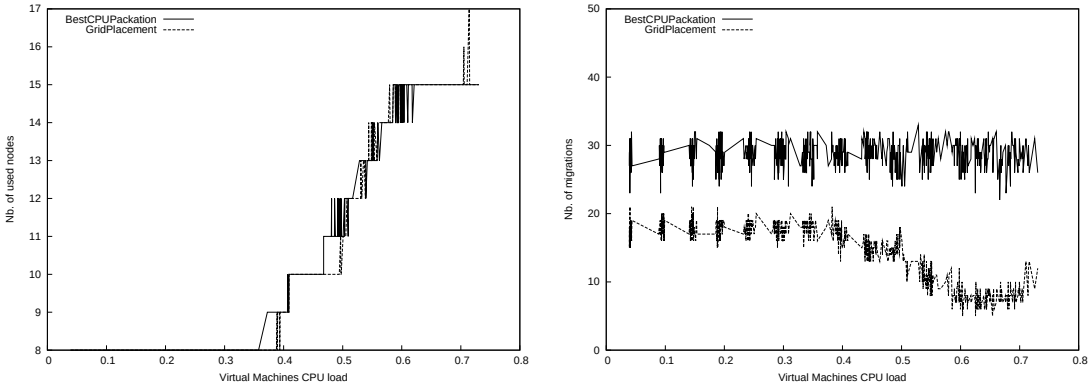
4.1. Comparaison entre les approches

Dans cette section, nous comparons une approche heuristique de type *BestFit*, une approche qui après évaluation est proche d'un compactage optimal, à l'approche complète utilisée dans Entropy. Cette comparaison tient compte à la fois de l'efficacité de l'algorithme (efficacité du compactage) mais également de la qualité de la configuration résultat (taille du plan de reconfiguration).

L'algorithme BestFit permet de compacter les machines virtuelles sur le minimum de nœuds possible. Les machines virtuelles les plus gourmandes en CPU sont placées en priorité sur les nœuds ayant la capacité CPU libre la plus petite, mais suffisante pour accueillir la machine virtuelle. On assure également que le nœud d'accueil dispose de suffisament de mémoire vive libre.

Afin de comparer ces deux approches, différentes configurations initiales (appelées *instances*) sont générées. Pour cette évaluation, les instances représentent l'état d'un cluster constitué de 20 machines mono-processeur de même capacité disposant chacune de 2Go de mémoire vive. Sur ces machines s'exécutent 30 machines virtuelles avec chacune 512Mo de mémoire vive. Pour chaque instance, nous faisons varier la consommation CPU de chacune des machines virtuelles afin d'étudier l'efficacité des algorithmes en fonction de la charge moyenne de celles-ci.

Nous remarquons d'après la figure 5(a) que les deux approches sont d'une efficacité comparable. En effet, le nombre de nœuds nécessaire à l'exécution des machines virtuelles est sensiblement le même. Des différences



(a) Efficacité du compactage (b) Nombre de migrations

FIG. 5 – Comparaison de l'heuristique *BestFit* avec Entropy

apparaissent cependant pour une charge moyenne comprise entre 40% et 50% puis autour de 60%. Dans le premier cas, Entropy réussit à améliorer le nombre de nœuds nécessaire d'une unité en trouvant la solution optimale. Dans le second, l'algorithme de Entropy est dans une transition de phase. Le temps qui lui est imparti pour trouver, améliorer ou prouver le résultat est alors insuffisant.

La seconde évaluation considère la qualité de la configuration générée. Dans cette situation, nous estimons la taille du plan de reconfiguration généré pour chaque approche. Dans le cas de l'approche de Entropy, le module d'optimisation de la reconfiguration est utilisé. La figure 5(b) nous montre que l'approche qualitative d'Entropy garantit un meilleur plan de reconfiguration qu'une approche heuristique classique et ce quelle que soit la charge moyenne des machines virtuelles. Ceci s'explique par l'utilisation du solveur de contraintes, disposant de mécanismes évolués permettant d'optimiser de manière itérative les plans de reconfiguration et de faire la preuve de l'optimalité du plan retourné. L'approche heuristique quant à elle ne fait que parcourir un ensemble de solutions, sans notions avancées d'amélioration. Dans ce cas, le temps imparti est insuffisant et ne permet pas de *découvrir* un bon plan de reconfiguration.

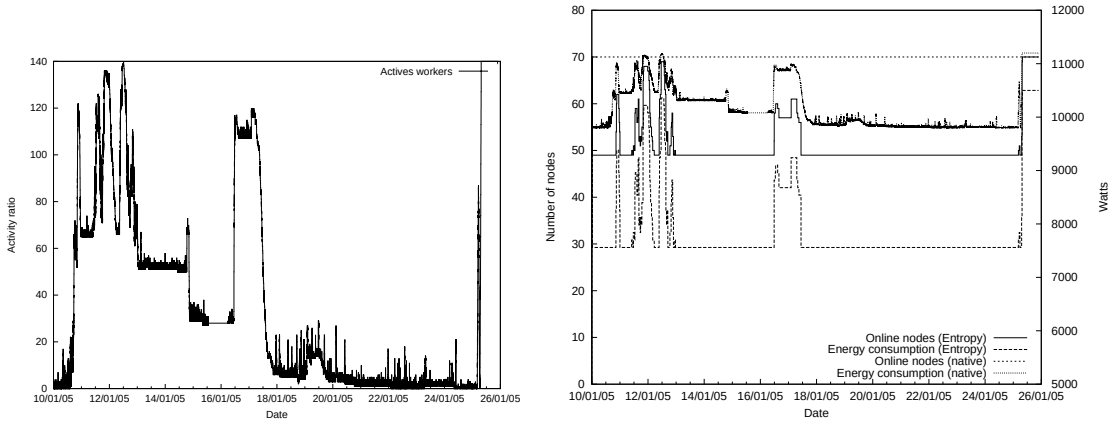
4.2. Évaluation d'après des traces réelles

Dans cette section, nous évaluons Entropy en utilisant des traces réelles d'exécution d'une application distribuée de calcul scientifique. L'application est une application de type maître/esclave où un nœud maître soumet des tâches à exécuter sur un ensemble de nœuds esclaves.

Un simulateur reproduit cet environnement composé de noeuds et de machines virtuelles, utilisable par Entropy. Une interface de contrôle reproduit le comportement du nœud maître et permet d'envoyer des tâches qui seront exécutées sur les machines virtuelles. Lorsqu'une tâche est exécutée, elle consomme la totalité des ressources CPU utilisables. Une horloge virtuelle permet d'accélérer ou de ralentir le temps afin de pouvoir exécuter de longues traces sur un intervalle de temps réduit. Les caractéristiques du cluster telle que la consommation électrique, le temps de migration des machines virtuelles ou le temps de démarrage des nœuds sont paramétrables par l'utilisateur.

La trace utilisée pour cette évaluation provient du cluster LPC de l'Université de Blaise-Pascal (France). Cette trace représente l'activité du cluster du 10 Janvier au 26 janvier 2005 (voir figure 6(a)). Ce cluster est composé de 70 machines bi-processeurs à base de Pentium-IV Xeon à 3GHz. Le nœud maître allouant les tâches à un processeur particulier, le cluster peut exécuter 140 tâches simultanément. Pour l'évaluation, chaque machine virtuelle est mono-processeur, le simulateur reproduit donc un cluster composé de 70 nœuds et 140 machines virtuelles pouvant s'exécuter simultanément. Les caractéristiques du cluster ont été définies à partir de mesures réalisées sur un cluster personnel à base de Sun Fire V20Z.

La figure 6(b) montre que le critère de concentration des machines virtuelles permet de réduire efficacement la consommation électrique en fonction des besoins de l'application. Pour cette simulation, Entropy permet une économie d'énergie allant de $6,1\%$ lorsque le cluster est utilisé à son maximum à $22,8\%$ lorsque le nombre de machines virtuelles active est au minimum.



(a) Activité du cluster (b) Comparaison énergétique

FIG. 6 – Évaluation dynamique de Entropy

Considérant `nbVMactives` et `nbVMrepos` comme respectivement le nombre de machines virtuelles actives et au repos puis $\texttt{max}(\texttt{nbVMactives}(\mathcal{H}_\texttt{i}))$ et $\texttt{max}(\texttt{nbVMrepos}(\mathcal{H}_\texttt{i}))$ comme le nombre maximum de machines virtuelles actives ou au repos hébergeables par un nœud $\mathcal{H}_\texttt{i}$, il est possible d'estimer le nombre de nœuds actifs à un moment donné pour ce type de cluster :

$$\mathrm{N} = \frac{\texttt{nbVMactives}}{\texttt{max}(\texttt{nbVMactives}(\mathcal{H}_\texttt{i}))} + \frac{\texttt{nbVMrepos}}{\texttt{max}(\texttt{nbVMrepos}(\mathcal{H}_\texttt{i}))} \tag{4}$$

Si l'on souhaite améliorer le critère de concentration d'Entropy pour cette situation sans modifier l'architecture matérielle du cluster, il importe d'améliorer le nombre maximum de machines virtuelles inactives sur un même nœud. Une machine virtuelle inactive ayant une consommation CPU négligeable, le critère à optimiser est donc la quantité de mémoire vive allouée à chaque machine virtuelle. Réduire cette quantité maximiserait la concentration de machines virtuelles inactives et réduirait donc le nombre de nœuds actifs. Il faut cependant tenir compte des besoins en mémoire des applications et du coût de la mise en place d'un système ré-allouant dynamiquement la mémoire vive des machines virtuelles.

4.3. Impact d'Entropy

En rejouant l'activité d'un cluster réel, nous pouvons estimer la perte de performance liée à notre solution en comparant les résultats d'exécution théorique (la durée du calcul des tâches) avec ceux d'Entropy. Sur le cluster réel, chaque tâche est exécutée sur un nœud actif et consomme 100% du processeur alloué. Sur Entropy, les machines virtuelles inactives sont concentrées afin d'économiser de l'énergie. Ainsi, si plusieurs tâches arrivent sur des machines virtuelles inactives et que ce nombre dépasse la capacité du nœud en machines virtuelles actives, alors le nœud devient surchargé et la durée d'exécution théorique des différentes tâches est augmentée (dû au partage des ressources processeurs).

En considérant respectivement $\mathcal{A}_\texttt{i}^{\texttt{th}}$ et $\mathcal{A}_\texttt{i}^{\texttt{reel}}$ les temps restant théorique et réel pour l'exécution d'une tâche sur la machine virtuelle $\texttt{i}$, et en considérant $\mathcal{H}_\texttt{i}$ le nœud hébergeant la machine virtuelle $\texttt{i}$, nous définissons un facteur de dégradation par la formule suivante :

$$\mathcal{D} = \frac{\sum \mathcal{A}_\texttt{i}^{\texttt{reel}}}{\sum \mathcal{A}_\texttt{i}^{\texttt{th}}} \tag{5}$$

$$\mathcal{A}_\texttt{i}^{\texttt{reel}} = \begin{cases} \mathcal{A}_\texttt{i}^{\texttt{th}}, \texttt{max}(\texttt{nbVMactives}(\mathcal{H}_\texttt{i})) \geq \texttt{nbVMactives}(\mathcal{H}_\texttt{i}) \\ \mathcal{A}_\texttt{i}^{\texttt{th}} \times \frac{\texttt{nbVMactives}(\mathcal{H}_\texttt{i})}{\texttt{max}(\texttt{nbVMactives}(\mathcal{H}_\texttt{i}))}, \text{sinon} \end{cases} \tag{6}$$

Un facteur de dégradation proche de 1 implique que les machines virtuelles actives sont rapidement migrées et isolées sur un nœud disposant de ressources processeur libres. La durée de la phase de calcul de la solution est donc acceptable et ne dégrade pas les performances du cluster. Un facteur de dégradation important implique à l'inverse que trop de machines virtuelles actives sont concentrées sur un nombre trop réduit de nœuds ; les machines virtuelles ne sont pas isolées assez rapidement et les performances du cluster se dégradent. La figure 7 résume l'évolution du facteur de dégradation durant la simulation.

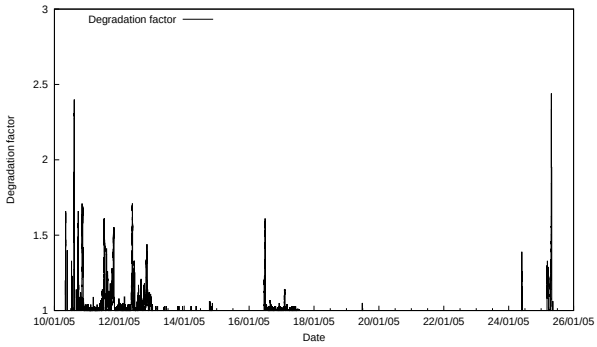


FIG. 7 – Évolution du facteur de dégradation

Cette figure montre que la dégradation est négligeable la plupart du temps. Cependant, nous observons une perte de performance significative lorsque le cluster subit une importante augmentation de l'activité (entre le 11 et le 13 janvier). Dans ce cas, le temps de calcul de la nouvelle configuration est important comparé au temps entre chaque reconfiguration. Les informations prélevées durant la phase de monitoring ne sont donc plus assez fiables et le placement est moins bon. Néanmoins notre évaluation montre que notre approche basée sur la recherche de solutions complètes est efficace. Considérer la taille des plans de reconfiguration comme un critère qualité important permet d'améliorer la réactivité d'Entropy. Il n'y a pas de phase de stagnation avec un facteur de dégradation supérieure à 1 : le solveur réussit à trouver une solution optimale dans un temps suffisant pour minimiser le nombre de nœuds allumés sans pour autant pénaliser le cluster au niveau de ses performances par rapport à l'exécution théorique.

5. État de l'art

5.1. Reconfiguration dynamique de grilles

L'adaptation est un problème récurrent dans le domaine des grilles de calcul. Ruth *et al.* [21] proposent une solution auto-adaptative au niveau des machines virtuelles. Un service d'adaptation centralisé, qui dispose d'une vue globale de la grille, re-affecte les différentes machines virtuelles en fonction de leurs besoins en ressources CPU. L'algorithme de placement est basé sur l'algorithme *BestFit* exposé précédemment et propose une concentration des machines virtuelles efficaces. Néanmoins, cette approche heuristique est limité à un seul critère et une extension à un système multi-critère implique une ré-ingéniérie complète de l'approche.

Othman *et al.* [18] proposent une approche utilisant un gestionnaire de ressource adaptatif. Celui-ci prend la décision de déplacer à la volée des tâches suspectées de fonctionner de manière dégradée sur des machines plus puissantes. La détection d'un comportement anormal est basée sur un système prédictif utilisant la supervision de l'exécution et des données décrivant les profils de tâches. La couche d'abstraction basée sur l'utilisation d'un intergiciel et de la réflexion implique une ré-ingéniérie des applications ainsi qu'un modèle de programmation précis. L'affectation des machines virtuelles, guidée par une heuristique, ne traite les tâches qu'une à une, les décisions peuvent donc avoir des effets de bords dégradant les performances d'autres composants.

Les systèmes à bases de règles permettent une plus grande flexibilité dans la définition des critères de placement. Dans Océano [3], un moteur de règles basé sur des contrats SLA assure la disponibilité d'une quantité de ressources correspondant aux besoins spécifiés dans les différents contrats. Ceux-ci définissent des valeurs seuil qui sont utilisées pour déclencher un événement de reconfiguration lorsqu'une violation est détectée. Un système de classe permet d'assigner un certains niveau de priorité à chaque utilisateur lorsque des conflits de règles se produisent.

Bien que ce type d'architecture permet un certain niveau de composition des critères, cette architecture n'empêche pas les conflits entre les règles ayant un certain recouvrement que ce soit sur la source ou la cible de la règle [15]. Dans notre contexte, ce recouvrement est possible : chaque déplacement de machines virtuelle à un impact sur plusieurs noeuds et différentes contraintes sélectionnées à la fois par un administrateurs et un utilisateur peuvent porter sur un même ensemble de machines virtuelles. L'approche PPC, par la notion de propagation, permet en théorie d'obtenir de meilleurs résultats que les moteurs à base de règles, tout en détectant et en empêchant les risques de conflits entre contraintes.

5.2. Gestion d'énergie

La consommation d'énergie des grilles est une contrainte importante limitant encore aujourd'hui leur expansion. Différents travaux se concentrant sur les grilles mais aussi sur les clusters tentent de résoudre ce problème. Les premières solutions considéraient le problème localement et adaptaient la tension d'alimentation de composants (CPU, disques dur) en fonction de la charge des machines (DVS). Dans le cas de systèmes distribués, où la distribution d'énergie est différente, les gains sont moins significatifs. Ceci s'explique en partie par l'existence de périphériques difficilement régulables comme les ventilateurs et consommant même au repos une quantité d'énergie importante. Une station Sun fire V20Z par exemple, consomme 140 watts au repos et 150 Watts lorsque cette machine est surchargée.

Les solutions pour systèmes distribués abordent le problème d'une façon globale. Les premiers travaux autour des clusters [19, 6] considèrent qu'il est nécessaire de concentrer l'activité sur un nombre limité de machines puis d'éteindre les machines non-utilisées. Cela peut être vu comme une reconfiguration où l'on concentre la charge en fonction des besoins. Les approches ont ensuite évolué pour combiner solutions locales et globales [9], et s'adapter aux environnements hétérogènes [12].

Notre critère de concentration des machines virtuelles s'inspire de ce type d'approche puisqu'en concentrant les machines virtuelles actives, nous concentrons également la charge. Notre approche diffère par contre par son aspect générique et transparent pour l'utilisateur. En effet, les solutions évoquées précédemment sont souvent implémentées au niveau applicatif, nécessite un modèle de distribution maître/esclave et une ré-ingéniérie des applications. Les tâches ne sont également pas migrées dynamiquement : l'activité de concentration est réalisée par le répartiteur de charge qui envoie seulement de nouvelles tâches sur un ensemble réduit de nœuds, limitant la portée de ces solutions à des systèmes ayant des tâches de courte durée (requêtes web par exemple).

Des solutions de reconfiguration utilisant une couche d'abstraction à base de virtualisation sont apparues récemment. Nathuji *et al.* [17] propose également un ensemble de règles permettant de coordonner stratégies locales et stratégies globales dans une architecture cluster virtualisée. Enfin, dans de précédents travaux [13], nous proposions une solution qui était à la base de Entropy. L'approche actuelle du critère d'économie d'énergie n'utilise plus une heuristique pour le placement mais une approche complète permettant un passage à l'échelle et une réactivité plus importante grâce à l'optimisation du temps de reconfiguration.

6. Conclusion et perspectives

Nous avons proposé dans cet article une architecture extensible permettant un placement dynamique des environnements utilisateurs à des fins d'optimisation. L'utilisation de la virtualisation permet d'encapsuler toutes les spécificités de chaque système dans différentes machines virtuelles. La supervision et la manipulation de ces environnements est alors transparente pour l'utilisateur et non-intrusive. Le moteur de reconfiguration quant à lui est construit à partir d'une approche de Programmation Par Contraintes et permet de spécialiser à la volée et en fonction des besoins les critères de placements et d'optimisation.

Notre prototype, Entropy, utilise actuellement des contraintes assurant la disponibilité des ressources en fonction des besoins, mais assure également un compactage des machines virtuelles minimisant le nombre de nœuds actifs. Nos évaluations, basées sur une comparaison avec une approche heuristique proche d'un placement optimale ou sur une trace réelle d'exécution d'application scientifique distribuée ont permit de valider notre approche. Les résultats bruts obtenus sont équivalent ou meilleurs à des solutions heuristiques. Cependant, l'utilisation d'une approche complète nous permet d'améliorer énormément la qualité des solutions obtenues, en minimisant les temps de reconfiguration.

Les perspectives concernant nos travaux portent d'abord le moteur de reconfiguration. Celui-ci n'est pas assez accessible aux utilisateurs et requiert un trop haut niveau d'expertise en PPC pour être facilement spécialisé. Une approche envisageable permettant de faciliter cette étape serait de proposer une abstraction langage : les utilisateurs

et administrateurs décrivent alors leurs architectures ainsi que les différentes contraintes dans un langage adapté qui serait ensuite transformées en un modèle PPC utilisable par le solveur de contraintes.
Il est nécessaire également d'approfondir l'approche actuelle de la supervision. Exprimer les besoins en ressources d'après l'analyse brute des machines virtuelles et sans implication de l'utilisateur n'apparaît viable que pour certains types d'applications comme des applications de calculs intensifs où l'expression des besoins est basique et facilement identifiable. D'autres types d'application comme les serveurs web ou les bases de données ont des besoins plus spécifiques qui ne peuvent être détectés efficacement avec une analyse de bas niveau. Une solution consiste alors à introduire une collaboration entre le système de supervision de l'hyperviseur et un système de plus haut niveau fournit par l'utilisateur et la reconfiguration

Bibliographie

1. Enabling grids for e-science, http ://www.eu-egee.org/.
2. Sumalatha Adabala, Vineet Chadha, Puneet Chawla, Renato Figueiredo, José Fortes, Ivan Krsul, Andrea Matsunaga, Mauricio Tsugawa, Jian Zhang, Ming Zhao, Liping Zhu, and Xiaomin Zhu. From virtualized resources to virtual computing grids : the in-vigo system. *Future Gener. Comput. Syst.*, 21(6) :896–909, 2005.
3. K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, and B. Rochwerger. Oceano-sla based management of a computing utility. *Proceedings of the Integrated Network Management International Symposium*, pages 855–868, 2001.
4. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 164–177, Bolton Landing, NY, USA, October 2003. ACM Press.
5. Frédéric Benhamou, Narendra Jussien, and Barry O'Sullivan, editors. *Trends in Constraint Programming*. ISTE, London, UK, May 2007.
6. Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *SOSP '01 : Proceedings of the eighteenth ACM Symposium on Operating Systems Principles*, pages 103–116, Banff, Alberta, Canada, 2001. ACM Press.
7. Choco solver, http ://choco-solver.net.
8. Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*, Boston, MA, USA, May 2005.
9. Mootaz Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient server clusters. In *Proceeding of the second Workshop on Power Aware Computing Systems*, pages 179–196, Cambridge, MA, USA, 2002.
10. Renato Figueiredo, Peter Dinda, and Jose Fortes. A case for grid computing on virtual machines. *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 550–559, 19-22 May 2003.
11. Ian T. Foster. The anatomy of the grid : Enabling scalable virtual organizations. In *Euro-Par '01 : Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, pages 1–4, London, UK, 2001. Springer-Verlag.
12. Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira Jr., and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *PPoPP '05 : Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 186–195, New York, NY, USA, 2005. ACM Press.
13. Fabien Hermenier, Nicolas Loriant, and Jean-Marc Menaud. Power management in grid computing with xen. In *Proceedings of 2006 on XEN in HPC Cluster and Grid Computing Environments (XHPC06)*, number 4331 in LNCS, pages 407–416, Sorento, Italy, 2006. Springer Verlag.
14. Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, 2003.
15. Emil C. Lupu and Morris Sloman. Conflicts in policy-based distributed systems management. *IEEE Trans. Softw. Eng.*, 25(6) :852–869, 1999.
16. Marvin McNett, Diwaker Gupta, Amin Vahdat, and Geoffrey M. Voelker. Usher : An Extensible Framework for Managing Clusters of Virtual Machines. In *Proceedings of the 21st Large Installation System Administration Conference (LISA)*, November 2007.
17. Ripal Nathuji and Karsten Schwan. Virtualpower : Coordinated power management in virtualizaed entreprise systems. In *Proc of the 21st Symposium on Operating Systems Principles (SOSP)*, october 2007.
18. Abdulla Othman, Peter Dew, Karim Djemame, and Iain Gourlay. Adaptive grid resource brokering. *cluster*,

00 :172, 2003.
19. Edouardo Pinheiro, Richardo Bianchini, Enrique Carrera, and Taliver Heath. Dynamic cluster reconfiguration for power and performance. In L. Benini, M. Kandemir, and J. Ramanujam, editors, *Compilers and Operating Systems for Low Power*. Kluwer Academic Publishers, 2002.
20. Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.
21. P. Ruth, Junghwan Rhee, Dongyan Xu, R. Kennell, and S. Goasguen. Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure. *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, pages 5–14, 2006.
22. Paul Shaw. A Constraint for Bin Packing. In *Principles and Practice of Constraint Programming (CP'04)*, volume 3258 of *Lecture Notes in Computer Science*, pages 648–662. Springer, 2004.
23. Parallel Workloads Archive - http ://www.cs.huji.ac.il/labs/parallel/workload/index.html.